CECOM

CENTER FOR SOFTWARE ENGINEERING

ADVANCED SOFTWARE TECHNOLOGY

Subject: **Final Report - Real-Time Ada Performance Benchmarks: Execution Results**

CIN: **C02 092LY 0004 00**

23 July 1990

REPORT DOCUMENTATION PAGE

Form Approved OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE 23 July 1990

3. REPORT TYPE AND DATES COVERED Final Report

4. TITLE AND SUBTITLE
Real-Time Ada Performance Benchmarks: Execution Results

5. FUNDING NUMBERS
F30602-86-C-0111

6. AUTHOR(S)
Arvind Goel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Unixpros Inc.
16 Birch Lane
Colts Neck, NJ 07722

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army HQ CECOM
Center for Software Engineering
Fort Monmouth, NJ 07703-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
STATEMENT A UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A set of real-time performance benchmarks were previously developed by CECOM to measure the performance of individual Ada features, to determine Ada runtime system implementation dependencies, and to determine the performance of paradigms found in real-time systems. This task transported the benchmarks to other environments and listed the results after running them. Using the DDC-I Ada cross compiler system hosted on a MicroVax II and targeted to the Intel 80X86 family of microprocessors, results were obtained. Also the results of running the benchmarks on a HP 9000/350 computer, using two versions of the self-hosted HP Ada compiler (HP3.25 and HP4.35) running under HP-UX, are presented.

14. SUBJECT TERMS
Ada, benchmarks, real-time

15. NUMBER OF PAGES 84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - DOD - Leave blank.
DOE - DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - NASA - Leave blank.
NTIS - NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Real-Time Ada Performance Benchmarks:

Execution Results

Prepared For:

U.S. Army CECOM
Advanced Software Technology
Center for Software Engineering
Fort Monmouth, NJ 07703-5000

Prepared By:

Unixpros Inc.
16 Birch Lane
Colts Neck, NJ 07722

July 10, 1990.

Real-time Ada Performance Benchmarks:

Execution Results

Executive Summary

This report documents the results from running the Real-time Ada Performance Benchmarks on the Intel 80386 computer using the DDC-I Ada compiler system hosted on a MicroVax II and targeted to Intel 8086 family of microprocessors. Results of running the benchmarks on a HP 9000/350 computer using the self-hosted HP Ada compiler running under HP-UX are also listed in the Appendix. A brief discussion of the benchmarks and test environment is followed by listing of the results obtained.

CONTENTS

LIST OF TABLES

1. Introduction

The Center for Software Engineering, Ft. Monmouth, NJ has been involved with developing benchmarks for Ada language and runtime features considered important for programming real-time embedded applications. As part of this effort, real-time performance benchmarks were developed that measure the performance of individual Ada features, determine Ada runtime system implementation dependencies, and determine the performance of paradigms found in real-time systems [1]. The report titled "Real-time Performance Benchmarks for Ada" should be consulted for an in-depth discussion of the benchmarks as well as the strategy used in designing the real-time benchmarks [1]. Report [1] also contains the results obtained by running the Real-time benchmarks on a Verdix Ada compiler system hosted on SUN 3/60 and targeted to a Motorola 68020 processor. In this report, the real-time benchmarks have been run on a Intel 80386 computer using the DDC-I Ada compiler system targeted to Intel 8086 microprocessor.

Section 2 briefly describes the Real-time benchmarks.

Section 3 describes the organization of the Real-time Benchmarks.

Section 4 lists the execution results obtained by running the benchmarks on a Intel 80386 computer.

Section 5 concludes with some thoughts about the results obtained during this effort.

Results of running the benchmarks on a HP 9000/350 computer using two versions of the self-hosted HP Ada compiler (namely HP3.25 and HP4.35) running under HP-UX are presented in Appendix A.

2. Real-time Benchmarks

The Real-time benchmarks were developed at the Advanced Software Technology, Center for Software Engineering, US Army CECOM. A detailed discussion of the Real-time benchmarks is presented in reference [1]. Also, the report titled "Documentation For Real-time Benchmarks For Ada" describes the organization, layout and file naming conventions of the Real-time benchmarks [2].

The Real-time benchmarks use three distinct approaches to determine performance of Ada compiler systems. These approaches are discussed briefly in the following sections.

2.1 Measure Performance Of Individual Features

This approach measures the execution speed of individual features of the language and runtime system by isolating the feature to be measured to the finest extent possible. Such benchmarks are useful in understanding the efficiency of a specific feature of an Ada implementation. For example, a benchmark that measures the time for a simple rendezvous can be run on two Ada compiler systems. Based on the results, an application can choose one compiler system over the other. The advantage of such an approach is performance evaluation without bias towards any application. These tests are useful for bottleneck analysis in which a score for a given test must exceed a stated threshold if an Ada implementation is to be considered suitable for an application. Benchmarks have been designed for Ada features such as tasking, memory management, exception handling, Chapter 13, interrupt Handling, CLOCK overhead and Type Duration, numeric computations, subprogram call overhead, Pragmas, and Input/Output.

2.2 Determining Runtime System Implementation

These benchmarks are concerned primarily with determining the implementation characteristics of an Ada Runtime System. The scheduling algorithm, storage allocation/deallocation algorithm, priority of rendezvous between two tasks without explicit priorities are some of the many implementation dependent characteristics that need to be known to determine if a compiler system is suitable for a particular real-time embedded application. Some implementation dependencies cannot be benchmarked and that information has to be obtained from the compiler vendor as well as the documentation supplied by the vendor. A major effort in such benchmarks involves interpreting the results obtained by running the benchmarks and drawing the correct conclusions. A detailed description has been provided to help interpret the results. The ARTEWG document "Catalog of Ada Runtime Implementation Dependencies" [4] lists those Ada features that are implementation dependent. This document has been consulted extensively in determining which implementation dependencies need to be benchmarked for real-time embedded systems. Benchmarks have been designed for Ada features such as: tasking, scheduling and delay statement, memory management, exceptions, interrupt handling, and asynchronous I/O.

2.3 Real-time Paradigms

This approach involves programming algorithms found in embedded systems. For example, a situation in real-time systems may be a producer that monitors a sensor and produces output asynchronously and sends it to a consumer. The producer task cannot wait for a rendezvous with the consumer (who might be doing something else) as the producer task might miss a sensor reading. To program this paradigm in Ada requires three tasks: a producer task, a buffer task that receives input from the producer task and sends the input to the third task:consumer task.

Macro constructs are defined as a set of Ada statements that perform a well defined process e.g. semaphores, mailbox construct etc. For real-time embedded systems, real-time paradigms can be identified and programmed in Ada using macro constructs. These benchmarks can be run on Ada compiler implementations and statistics gathered on their performance.

3. Organization of the Real-time Benchmarks

This section describes the organization of the Real-time benchmarks. For more details refer to the report "Documentation For Real-time Performance Benchmarks for Ada" [2].

At the top level of the directory structure is the directory **bench**. There are 3 directories under **bench**:

- **micro**: The directory micro contains the benchmarks that measure the performance of Ada features that are important for real-time embedded applications.
- **rts**: The directory rts contains benchmarks that measure runtime implementation dependencies.
- **paradigms**: The directory paradigms contains benchmarks that implement macro constructs and real-time programming paradigms.

3.1 Micro

The directory micro contains the directories that contain the micro benchmarks.

- **lv** lv contains loop verification benchmarks. This benchmark is executed to verify that textually similar loops should take equal amount of time to execute.
- **t**: t contains tasking activation/termination benchmarks. These benchmarks measure tasking activation/termination timings under various conditions.
- **r**: r contains tasking synchronization benchmarks. These benchmarks determine the time required to perform rendezvous under various loads and conditions.
- **ex**: ex contains exception handling related benchmarks. Exception handling and propagation timings are measured by these benchmarks.
- **chap13**: chap13 contains Chapter 13 related benchmarks. These benchmarks may not compile for some Ada compiler systems.
- **dd**: dd contains benchmarks that measure dynamic allocation time in declarative regions.
- **dn**: dn contains dynamic allocation with new operator benchmarks (file dn_compile contains the list of benchmarks that need to be compiled for these

cases). The directory dn also contains benchmarks that determine allocation time without memory being freed by UNCHECKED_DEALLOCATION (file dn_compile1 contains the list of files that need to be compiled for these benchmarks). The directory **new** under dn contains tests that determine the affect of additional tasks on time for dynamic allocation (file dn_compile contains files for 5 tasks and dn_compile1 contains for 10 tasks).

- **co**: co contains Clock function calling overhead and resolution benchmarks.
- **io**: io contains input/output benchmarks.
- **tm**: tm contains mathematical benchmarks.
- **d**: d contains regular case do nothing subprogram overhead benchmarks.
- **i**: i contains inline subprogram overhead benchmarks.
- **p**: g contains cross package subprogram overhead benchmarks.
- **g**: g contains generic subprogram overhead benchmarks.
- **c**: c contains generic cross package subprogram overhead benchmarks.
- **dt**: dt contains scheduling and delay statement benchmarks.

3.2 rts

The directory rts under bench contains the benchmarks that determine runtime implementation dependencies. The following directories exist under rts:

- **t**: t contains benchmarks that determine tasking related runtime implementation dependencies.
- **r**: r contains benchmarks that determine rendezvous related runtime implementation dependencies.
- **mm**: mm contains benchmarks that determine memory management related runtime implementation dependencies.
- **ex**: ex contains benchmarks that determine exception related runtime implementation dependencies.

3.3 paradigms

The directory paradigms contains the benchmarks that determine the performance of macro constructs and real-time paradigms.

4. Benchmark Execution on DDC-I Ada Compiler

This section presents the results of running the Real-time benchmarks on the DDC-I Ada Compiler System targeted to the Intel 8086 computer. The benchmarks have been run on a Intel 80386 computer (as Intel 8086 code can also run on the Intel 80386).

4.1 Testbed Hardware and Software

The hardware used was MicroVAX II running MicroVMS 4.2, linked to a Tandy Corporation Intel 80386 computer.

Host: **MicroVAX II, running MicroVMS 4.2.**

Compiler: **DDC-I Ada Development System targeted to Intel 8086 target**

Target: **Intel 80386 computer with 1 megabyte of RAM.**

The benchmarks were compiled on the MicroVAX II and then downloaded to the Intel 80386 computer via Kermit. The benchmarks were compiled without the optimize option and the timings listed are for un-optimized runs.

4.2 Microscopic Benchmarks

4.2.1 Tasking

4.2.1.1 Task Activation/Termination

Table 1 lists the benchmarks that have been developed for Task activation/termination.

TABLE 1. Task/Activation Termination Benchmarks

Execution time in milliseconds

File Name	Benchmark Description	Time
t00001.a	Task type in main, object in block statement	0.305
t00001_1	Task object is declared directly in block statement	0.366.8
t00001_2.a	Task type and object defined in package procedure	0.366
t00001_3.a	Task type in package, object in package procedure	0.366
t00001_4.a	Task type and object are declared in another task	0.356
t00002.a	Task type and array elaborated in a procedure	1.5
t00002_1.a	Task type in package, array in procedure	1.53
t00002_2.a	Task type in main, array in package procedure	1.53
t00003.a	Task object is declared as part of record	0.366
t00004.a	Task access type in main, task created via new	0.370
t00004_1.a	Task access type in block, task created via new	0.370
t00004_2.a	Task access type in main, array created via new	0.370
t00005.a	Task object in block statement, idle tasks=1	0.340
t00005_1.a	Task object in block statement, idle tasks=5	0.340
t00005_2.a	Task object in block statement, idle tasks=10	0.340
t00005_3.a	Task object in block statement, idle tasks=20	0.341
t00006.a	Task created via new allocator, idle tasks=1	0.370
t00006_1.a	Task created via new allocator, idle tasks=5	0.370
t00006_2.a	Task created via new allocator, idle tasks=10	0.370
t00006_3.a	Task created via new allocator, idle tasks=20	0.371

Some observations about the results in Table 1 are:

1. For the DDC-I compiler, the average time for task activation/termination for tasks declared in arrays is around 1.5 milliseconds, which is significantly higher than the task activation/termination time (0.35 milliseconds) for tasks declared in the main program. This is due to the fact that as each task in the array is elaborated, the task space for that task is left intact till all tasks in the array have been elaborated. Storage allocation times for tasks may deteriorate as more and more space has been allocated.

2. The DDC-I compiler takes more time (0.37 milliseconds) for task activation/termination timing via the new allocator as compared to task objects declared in the main program (0.35 milliseconds).

3. There is no effect on task activation/termination timings when the number of idle tasks increases.

4.2.1.2 Task Synchronization

Table 2 lists the benchmarks for simple rendezvous.

TABLE 2. Simple Rendezvous Benchmarks (No Parameters Passed)

Execution time in microseconds

File Name	Benchmark Description	Time
r00001.a	Procedure calls entry of task declared in main	67
r00001_1.a	Procedure calls entry in task created via new	67
r00001_2.a	Main calls entry in task decl in package	104
r00002.a	Main calls two entries in two tasks decl in package	103
r00002_1.a	Main calls 10 entries in ten tasks decl in package	102
r00002_2.a	Main calls 10 entries in one task decl in package	102
r00003.a	Main calls 1st entry in select, 2 entries decl	157
r00003_1.a	Main calls last entry in select, 2 entries decl	154
r00003_2.a	Main calls 1st entry in select, 10 entries decl	241
r00003_3.a	Main calls last entry in select, 10 entries decl	210
r00003_4.a	Main calls 6th entry in select, 10 entries decl	225
r00003_5.a	Main calls 1st entry in select, 20 entries decl	342
r00003_6.a	Main calls last entry in select, 20 entries decl	283
r00003_7.a	Main calls 11th entry in select, 20 entries decl	310
r00004.a	Main calls 1st entry out of 2, 1st guard true next false	150
r00004_1.a	Main calls last entry out of 2, 1st guard false next true	152
r00004_2.a	Main calls 1st entry out of 20, 1st guard true rest false	187
r00004_3.a	Main calls last entry out of 20, last guard true rest false	189
r00004_4.a	Main calls 11th entry out of 20, 11th guard true rest false	187
r00004_5.a	Main calls 11th entry out of 20, all guards true	339

Some observations from the results in Table 2 are:

1. Task rendezvous time for single entry calls in select is 67 microseconds. For tasks that are declared in packages and entry call made from the main program, the rendezvous time increases to 104 microseconds.

2. For the DDC-I compiler, the timing for rendezvous is nearly the same for the scenarios in which the main program calls ten entries in 10 different tasks or the main program calls 10 entries in one task.

3. The measurements indicate that the more the number of entries in a select statement, the more time it takes to rendezvous with any entry in the select statement.

4. Also, for the DDC-I compiler the later the position of the accept in the select statement, the less time it takes for the rendezvous to complete (without the guard statement).

5. The DDC-I compiler evaluates the guards before the entry call is made and hence guards have minimal effect on rendezvous time.

Table 3 lists complex rendezvous benchmarks.

TABLE 3. Complex Rendezvous Benchmarks

Execution time in microseconds

File Name	Direction Passed	Type and Number Passed	Size	Time (us)
r00005_i.a	In	Integer Array	1	107
r00005_o.a	Out	Integer Array	1	106
r00005_io.a	In Out	Integer Array	1	109
r00005_1_i.a	In	Integer Array	1000	108
r00005_1_o.a	Out	Integer Array	1000	110
r00005_1_io.a	In Out	Integer Array	1000	110
r00005_2_i.a	In	Integer Array	10000	107
r00005_2_o.a	Out	Integer Array	10000	109
r00005_2_io.a	In Out	Integer Array	10000	110
r00005_3_i.a	In	1 Integer		107
r00005_3_o.a	Out	1 Integer		110
r00005_3_io.a	In Out	1 Integer		108
r00005_4_i.a	In	10 Integers		168
r00005_4_o.a	Out	10 Integers		189
r00005_4_io.a	In Out	10 Integers		195
r00005_5_i.a	In	100 Integers		488
r00005_5_o.a	Out	100 Integers		627
r00005_5_io.a	In Out	100 Integers		663

Some observations about the results in Table 3 are:

1. The measurements indicate that integer arrays are passed by reference rather than by copy as the rendezvous time for integer arrays of 1 and 1000 are essentially the same.

2. As far as integer parameters are concerned, the DDC-I compiler uses pass by copy (due to the fact that the time for rendezvous increases with the increase in the number of integer parameters).

3. Also, the time for mode **out** and **in out** parameters is more than the time required for parameters of mode **in.** This is logical since the compiler has to copy back the change in value that can occur with a variable of type **out** or **in out.**

Table 4 lists more rendezvous benchmarks.

TABLE 4. More Rendezvous Benchmarks

Execution time in microseconds

File Name	Benchmark Description	Time (us)
r00006_1_1.a	1st entry out of 2 called with 10 integers	254
r00006_1_2.a	1st entry out of 2 called with 100 integers	737
r00006_2_1.a	Last entry out of 2 called with 10 integers	247
r00006_2_2.a	Last entry out of 2 called with 100 integers	719
r00006_3_1.a	1st entry out of 10 called with 10 integers	338
r00006_3_2.a	1st entry out of 10 called with 100 integers	806
r00006_4_1.a	Last entry out of 10 called with 10 integers	307
r00006_4_2.a	Last entry out of 10 called with 100 integers	785
r00006_5_1.a	1st entry out of 20 called with 10 integers	437
r00006_5_2.a	1st entry out of 20 called with 100 integers	906
r00006_6_1.a	Last entry out of 20 called with 10 integers	378
r00006_6_2.a	Last entry out of 20 called with 100 integers	864
r00007.a	Overhead due to terminate alternative	11
r00008.a	Overhead of conditional entry call,rendezvous complete	5
r00008_1.a	Overhead of conditional entry call,rendezvous incomplete	25
r00009.a	Overhead of timed entry call,rendezvous complete	5
r00009_1.a	Overhead of timed entry call,rendezvous incomplete	26
r00011.a	Main calls an entry with 100 Integers,Idle tasks = 1	665
r00011_1.a	Main calls entry with 100 Integers,Idle tasks = 5	665
r00011_2.a	Main calls entry with 100 Integers,Idle tasks = 10	665
r00011_3.a	Main calls entry with 100 Integers,Idle tasks = 20	665

Some observations about the results in Table 4 are:

1. For the DDC-I compiler, the time for rendezvous call to the last entry with 100 integer parameters (mode in out) increases from 719 microseconds (2 entries) to 785 microseconds (10 entries) to 864 microseconds (20 entries). Thus, it can be deduced that time for rendezvous with integer parameters increases linearly as the number of accept statements in the select statement increases.

2. For the DDC-I compiler time for rendezvous remains the same for up to 20 idle tasks.

4.2.2 Memory Management

Table 5 lists Dynamic allocation benchmarks when the storage allocated is fixed.

TABLE 5. Dynamic Allocation:Storage Allocated Is Fixed

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	Time (us)
dd_in1.a	Integer	1		0.3
dd_in10.a	Integer	10		0.3
dd_in100.a	Integer	100		0.3
dd_st1.a	String	1	1	0.3
dd_st10.a	String	1	10	0.3
dd_st100.a	String	1	100	0.3
dd_en1.a	Enumeration	1		0.4
dd_en10.a	Enumeration	10		0.4
dd_en100.a	Enumeration	100		0.4
dd_ar1.a	Array of Integer	1	1	0.3
dd_ar10.a	Array of Integer	1	10	0.3
dd_ar100.a	Array of Integer	1	100	0.3
dd_ar1k.a	Array of Integer	1	1000	0.4
dd_ar10k.a	Array of Integer	1	10000	0.4
dd_ar100k.a	Array of Integer	1	100000	0.4
dd_rc1.a	Record of Integer	1	1	0.3
dd_rc10.a	Record of Integer	1	10	0.3
dd_rc100.a	Record of Integer	1	100	0.3

Some observations about the results in Table 5 are:

1. For the DDC-I Compiler, time required to allocate integer variables, enumeration variables, strings and arrays of integers upon entering a subprogram was small (< 1 microsecond).

Table 6 lists dynamic allocation benchmarks when the storage allocated is variable.

TABLE 6. Dynamic Allocation:Storage Allocated Is Variable

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	Time (us)
dd_1d1.a	1-D Dynamically Bounded Array	1	1	12
dd_1d10.a	1-D Dynamically Bounded Array	1	10	12
dd_2d1.a	2-D Dynamically Bounded Array	1	1	26
dd_2d10.a	2-D Dynamically Bounded Array	1	100	26
dd_3d1.a	3-D Dynamically Bounded Array	1	1	45
dd_3d10.a	3-D Dynamically Bounded Array	1	1000	45

Some observations about the results in Table 6 are:

1. The time required for dynamically bounded arrays increases as the dimensions of the dynamically bounded array increase but not when the size of the object increases.

Table 7 lists dynamic allocation benchmarks with new allocator.

TABLE 7. Dynamic Allocation with NEW Allocator

Execution time in microseconds

File Name	Type Declared	Size of Object	Time (us)
dn_in1.a	Integer	1	102
dn_en1.a	Enumeration	1	101
dn_st1.a	String	1	101
dn_st10.a	String	10	101
dn_st100.a	String	100	101
dn_ar1.a	Integer Array	1	101
dn_ar10.a	Integer Array	10	101
dn_ar100.a	Integer Array	100	101
dn_ar1k.a	Integer Array	1000	190
dn_rc1.a	Record of Integer	1	101
dn_rc10.a	Record of Integer	10	101
dn_rc20.a	Record of Integer	20	101
dn_rc50.a	Record of Integer	50	101
dn_rc100.a	Record of Integer	100	101
dn_1d1.a	1-D Dynamically Bounded Array	1	203
dn_1d10.a	1-D Dynamically Bounded Array	10	203
dn_2d1.a	2-D Dynamically Bounded Array	1	211
dn_2d10.a	2-D Dynamically Bounded Array	100	211
dn_3d1.a	3-D Dynamically Bounded Array	1	238
dn_3d10.a	3-D Dynamically Bounded Array	1000	333

Observations that can be made from the results in Table 7 are:

1. Time to allocate a discrete variable via the new allocator is around 101 microseconds.
2. Time to allocate integer array of size 1000 or more is around 190 microseconds.
3. Time to allocate dynamically bounded arrays increases as the dimensions of the array increase.

Table 8 lists dynamic allocation benchmarks with new allocator and no storage deallocation takes place. In Table 8, the column **size of object** for strings is STRING'LENGTH, for integer arrays **size of object** is array'length, and for records the **size of object** is specified as the number of fields in the record.

TABLE 8. NEW Allocator:No Storage Deallocation

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	Time (us)
dn_in1.a	Integer	1	1	50
dn_en1.a	Enumeration	1	1	49
dn_st1.a	String	1	1	48
dn_st10.a	String	1	10	49
dn_st100.a	String	1	100	51
dn_ar1.a	Integer Array	1	1	48
dn_ar10.a	Integer Array	1	10	51
dn_ar100.a	Integer Array	1	100	51
dn_ar1k.a	Integer Array	1	1000	90
dn_rc1.a	Record of Integer	1	1	48
dn_rc10.a	Record of Integer	1	10	51
dn_rc20.a	Record of Integer	1	20	58
dn_rc50.a	Record of Integer	1	50	59
dn_rc100.a	Record of Integer	1	100	59
dn_1d1.a	1-D Dynamically Bounded Array	1	1	110
dn_1d10.a	1-D Dynamically Bounded Array	1	10	115
dn_2d1.a	2-D Dynamically Bounded Array	1	1	119
dn_2d10.a	2-D Dynamically Bounded Array	1	100	121
dn_3d1.a	3-D Dynamically Bounded Array	1	1	149
dn_3d10.a	3-D Dynamically Bounded Array	1	1000	151

Some observations about the results in Table 8 are:

1. For objects upto size 100 or less, time for dynamic memory allocation remains essentially the same, but for objects of size 1000 or more, memory allocation increases by about 80% or more.
2. In these test cases, memory allocated is not being freed and therefore, the measurements exclude the time to free the memory that is being allocated. Hence, the timings listed in Table 8 are less than the timings listed in Table 7.

Table 9 lists dynamic allocation benchmarks with new allocator when active tasks is 5. In Table 9, the column **size of object** for strings is STRING'LENGTH, for integer arrays **size of object** is array'length, and for records the **size of object** is specified as the number of fields in the record.

TABLE 9. NEW Allocator:Active Tasks = 5

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	Time (us)
dn_st100.a	String	1	100	101
dn_ar1k.a	Integer Array	1	1000	197
dn_rc100.a	Record of Integer	1	100	99
dn_1d10.a	1-D Dynamically Bounded Array	1	10	203
dn_2d10.a	2-D Dynamically Bounded Array	1	100	211
dn_3d10.a	3-D Dynamically Bounded Array	1	1000	333

Table 10 lists dynamic allocation benchmarks with new allocator when active tasks is 10 (object sizes are the same in Tables 9 and 10).. In Table 10, the column **size of object** for strings is STRING'LENGTH, for integer arrays **size of object** is array'length, and for records the **size of object** is specified as the number of fields in the record.

TABLE 10. NEW Allocator:Active Tasks = 10

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	Time (us)
dn_st100.a	String	1	100	101
dn_ar1k.a	Integer Array	1	1000	198
dn_rc100.a	Record of Integer	1	100	99
dn_1d10.a	1-D Dynamically Bounded Array	1	10	202
dn_2d10.a	2-D Dynamically Bounded Array	1	100	211
dn_3d10.a	3-D Dynamically Bounded Array	1	1000	333

Tables 9 and 10 show negligible impact of existing tasks in the system on the time for memory allocation/deallocation.

4.2.3 Exceptions

Table 11 below gives the results for exception handling times for exceptions raised and handled in a block for the DDC-I compiler. In this table, the word explicit has been used for exceptions raised via the **raise** statement, and implicit is used for abnormal conditions in the code.

TABLE 11. Exception Raised and Handled in Block

Execution time in microseconds

File Name	Exception not raised	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00001.a	0.2	92	117	125	122	131
e00001_1.a	0.2	108	133	140	135	146
e00001_2.a	0.2	134	158	166	162	173

Some observations from Table 11 are:

1. For the DDC-I compiler, the overhead associated with the code sequence (that has an exception handler associated with it, yet no exception is raised during the execution of that code) is negligible.

2. For the user-defined exception, exception handling times are much less than exception handling times for other exceptions.

3. As expected, times for handling NUMERIC_ERROR (implicitly raised) is higher than exception handling times for other exceptions. Exception handling times are increased as more tasks are active in the system.

Table 12 below gives the results (for DDC-I compiler) for exception handling times for exceptions raised and handled one level above: In this table, the word explicit has been used for exceptions raised via the **raise** statement, and implicit is used for abnormal conditions in the code.

TABLE 12. Exception Raised and Handled One Level Above

Execution time in microseconds

File Name	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00002.a	143	167	175	171	182
e00002_1.a	165	190	198	194	205
e00002_2.a	177	200	208	202	214

Some observations from Table 12 are:

1. After subtracting the timings obtained in the previous Table, it takes roughly about 22 more microseconds to propagate and handle the exception one level above where it is raised.

Table 13 below gives the results (for DDC-I compiler) for exception handling times for exceptions raised and handled more than one level above:

TABLE 13. More Exception Handling Benchmarks

Execution time in microseconds

File Name	Benchmark Description	Time
e00003.a	User Exception handled 3 procs above	241
e00003_1.a	User Exception handled 3 procs above,5 idle tasks	264
e00003_2.a	User Exception handled 3 procs above,10 idle tasks	275
e00004.a	User Exception Raised handled 4 procs above	290
e00004_1.a	User Exception handled 4 procs above,5 idle tasks	312
e00004_2.a	User Exception handled 4 procs above,10 idle tasks	324

Some observations from Table 13 are:

1. This benchmark reinforces the results about the extra time for each level (~50 microseconds: this is obtained by subtracting the time for e00003.a from e000004.a) that the exception has to be propagated.

Table 14 lists TASKING_ERROR exception benchmarks.

TABLE 14. Tasking_Error Exception Benchmarks

Execution time in microseconds

File Name	Benchmark Description	Time (us)
e00005.a	Exception Raised in rendezvous,0 idle tasks	115
e00005_1.a	Exception Raised in rendezvous,5 idle tasks	113
e00005_2.a	Exception Raised in rendezvous,10 idle tasks	113
e00006.a	Child task has error during elaboration,0 idle tasks	STORAGE_ERROR
e00006_1.a	Child task has error during elaboration,5 idle tasks	STORAGE_ERROR
e00006_2.a	Child task has error during elaboration,10 idle tasks	STORAGE_ERROR

Active tasks in the system have no effect on exceptions raised and handled during a rendezvous.

4.2.4 Chapter 13 Benchmarks

Table 15 lists all the Chapter 13 benchmarks.

TABLE 15. Chapter 13 Benchmarks

Execution time in microseconds

File Name	Benchmark Description	Time (us)
h00001.a	Boolean operations on arrays,Pragma PACK	10
h00001_1.a	Boolean operations on arrays,Rep Clause	Not compiled
h00001_2.a	Boolean operations on arrays,not packed	923
h00002.a	Boolean operations on array components,Pragma Pack	636
h00002_1.a	Boolean operations on array components,Rep Clause	Not compiled
h00002_2.a	Boolean operations on array components,not packed	157
h00003.a	Assignment,comparison on arrays of booleans,Pragma PACK	771
h00003_1.a	Assignment,comparison on boolean arrays,Rep Clause	Not compiled
h00003_2.a	Assignment,comparison on boolean arrays,not packed	24
h00004.a	Assign,compare whole records,no rep clause	11
h00004_1.a	Assign, compare whole records,rep clause	6
h00004_2.a	Assign, compare whole records,Pragma PACK	2
h00005.a	UNCHECKED_CONVERSION, INTEGER object to another	0.3
h00005_1.a	UNCHECKED_CONVERSION, STRING to INTEGER	2
h00005_2.a	UNCHECKED_CONVERSION,floating array to record	No response
h00006.a	Store, extract record bit fields, no rep clause	20
h00006_1.a	Store, extract record bit fields, rep clause	30
h00006_2.a	Store, extract record bit fields, rep clause	31
h00008.a	Store, extract record bit fields defined by nested rep clauses using packed arrays	Not compiled
h00009.a	Change of representation from one record to another	Not compiled
h00010.a	POS,SUCC, and PRED operations on enum type with rep clause numbered with gaps in internal coding	53
h00010_1.a	POS,SUCC, and PRED operations on enum type with rep clause numbered with no gaps in internal coding	54

Some observations about the results in Table 15 are:

1. Boolean operations on whole unpacked arrays is considerably higher than on whole packed arrays.
2. Time for boolean operations on packed array components is considerably higher than time for boolean operations on unpacked array components. The logical explanation is that it takes more time to unpack the record in order to perform the operation.
3. Time for Unchecked_Conversion is nearly zero showing good optimization by the compiler.
4. There is no effect on the execution time for enumeration representation clause with no gaps and enumeration type representation clause with gaps.
5. Some programs could not be compiled. The compiier vendor has been contacted.

4.2.5 Clock Function and TYPE Duration

Table 16 lists all the CLOCK tests.

TABLE 16. CLOCK Function Tests

Execution time in microseconds

File Name	Benchmark Description	Time (us)
c00001.a	CLOCK function overhead	607.59
c00002.a	CLOCK resolution	0.0001

Some observations about the results in Table 16 are:

1. For real-time applications, an overhead of 607 microseconds could be very time-expensive. Generally speaking, a CLOCK function overhead of 100 microseconds is more suitable for real-time applications. It has to be compared with the CLOCK function overhead of other Ada compilers.
2. The CLOCK resolution of 0.0001 microseconds is acceptable for real-time applications. Again, it has to be compared with the CLOCK resolution of other

Ada compilers.

4.2.6 Numeric Computation

Table 17 lists the benchmarks that calculate the overhead involved in dynamic computation of values of type TIME and DURATION.

TABLE 17. TIME and DURATION Mathematics

Execution time in microseconds

File Name	Operation Performed	Time (us)
tm1.a	Time = Var_time + Var_duration	263
tm2.a	Time = Var_time + Const_duration	263
tm3.a	Time = Var_duration + Var_time	263
tm4.a	Time = Const_duration + Var_time	263
tm5.a	Time = Var_time - Var_duration	262
tm6.a	Time = Var_time - Const_duration	262
tm7.a	Duration = Var_time - Var_time	22
tm8.a	Duration = Var_duration + var_duration	1.1
tm9.a	Duration = Var_duration + Const_duration	1.1
tm10.a	Duration = Const_duration + Var_duration	1.2
tm11.a	Duration = Const_duration + Const_duration	1.1
tm12.a	Duration = Var_duration - Var_duration	1.3
tm13.a	Duration = Var_duration - Const_duration	1.1
tm14.a	Duration = Const_duration - Var_duration	1.2
tm15.a	Duration = Const_duration - Const_duration	1.1

These timings have to be compared to results from other compilers.

Table 18 lists the results of running mathematical computation benchmarks.

TABLE 18. Numeric Computation Benchmarks

Execution time in microseconds

File Name	Operation Performed	Time
tm16.a	Float Matrix Multiplication	798.0
tm17.a	Float Matrix Addition	769.70
tm18.a	Factorial Calculation	89.0
tm19.a	Square root calculation	345.5

These timings have to be compared to results from other compilers.

4.2.7 Subprogram Overhead

Table 19 lists the types and modes of the parameters that are used in intra-package subprogram overhead tests and also lists the results. In Table 19, the headings under the Time column: I, O, I_O have the times listed for parameters with mode **in**, **out**, and **in out**.

TABLE 19. Subprogram Overhead (Intra-Package)

Execution time in microseconds

File Name	Type of Parameter Passed	Number Passed	Size	Time (us)		
				I	O	I_O
d_n.a		0	0	19		
d_i_1.a	Integer	1		9	19	19
d_i_10.a	Integer	10		19	29	29
d_i_100.a	Integer	100		49	68	117
d_e_1.a	Enumeration	1		9	19	19
d_e_10.a	Enumeration	10		19	29	29
d_e_100.a	Enumeration	100		48	78	107
d_a_1.a	Array of Integer	1	1	9	19	19
d_a_10.a	Array of Integer	1	10	9	19	19
d_a_100.a	Array of Integer	1	100	9	19	19
d_a_10k.a	Array of Integer	1	10000	9	19	19
d_r_1.a	Record of Integer	1	1	9	19	19
d_r_100.a	Record of Integer	1	100	19	19	19
d_u_a_1.a	Unconstrained array	1	1	19	19	19
d_u_a_100.a	Unconstrained array	1	100	9	19	19
d_u_a_10k.a	Unconstrained array	1	10000	9	19	19
d_u_r_1.a	Unconstrained record	1	1	9	19	19
d_u_r_100.a	Unconstrained record	1	100	9	19	19

Observations about the results in Table 19 are:

1. For integer and enumeration types, subprogram overhead for variables of mode **out** and **in out** is greater than that of mode **in**. This is because of the additional overhead involved in copying back the parameters of mode **out** and **in out** when returning from the procedure call.

2. Also, the overhead for passing 100 integers is higher than the overhead for passing 1 integer (due to the time required for copying the integers on the stack when the procedure call is made).

3. The results for arrays and records indicate that they are passed by reference as opposed to pass by copy.

4. The timings for unconstrained types suggest that there is very little extra overhead in passing the constraint information in the procedure call.

Table 20 lists the types and modes of the parameters that are used in intra-package tests with Pragma INLINE to determine if the INLINE pragma is supported and if it is, the amount of overhead involved in executing code generated by by an in-line expansion as opposed to executing the same set of statements originally coded without a subprogram call. In Table 20, the headings under the Time column: I, O, I_O have the times listed for parameters with mode **in, out,** and **in out.**

TABLE 20. Subprogram Overhead (Intra-Package with Pragma INLINE)

Execution time in microseconds

File Name	Type of Parameter Passed	Number Passed	Size	Time (us)		
				I	O	I_O
i_n.a		0	0	2		
i_i_1.a	Integer	1		2	3	4
i_i_10.a	Integer	10		2	4	4
i_i_100.a	Integer	100		8	8	10
i_e_1.a	Enumeration	1		2	3	3
i_e_10.a	Enumeration	10		2	4	4
i_e_100.a	Enumeration	100		7	10	9
i_a_1.a	Array of Integer	1	1	2	4	4
i_a_10.a	Array of Integer	1	10	2	4	4
i_a_100.a	Array of Integer	1	100	2	4	3
i_a_10k.a	Array of Integer	1	10000	2	4	4
i_r_1.a	Record of Integer	1	1	2	4	4
i_r_100.a	Record of Integer	1	100	4	4	4
i_u_a_1.a	Unconstrained array	1	1	4	4	4
i_u_a_100.a	Unconstrained array	1	100	2	4	4
i_u_a_10k.a	Unconstrained array	1	10000	2	4	4
i_u_r_1.a	Unconstrained record	1	1	2	4	4
i_u_r_100.a	Unconstrained record	1	100	2	4	3

Observation about the results in Table 20 are:

1. The overhead due to INLINE expansion of code for parameters of type integer and enumeration indicates that the overhead due to INLINE expansion is higher than the time it takes to execute the same set of statements without a procedure call.

2. For composite and unconstrained types, the timings indicate that the overhead in executing code produced by pragma INLINE is negligible.

Inter-Package Reference Tests: In inter-package reference, the calling subprogram is in a package other than the one in which the called subprogram resides. The motivation for inter-package tests is to compare the subprogram call overhead time between intra- and inter-package calls.

Table 21 lists the types of the parameters that are used in these tests and also lists the results for the DDC-I compiler. In Table 21, the headings under the Time column: I, O, I_O have the times listed for parameters with mode in, out, and in out respectively.

TABLE 21. Subprogram Overhead (Inter-Package)

Execution time in microseconds

File Name	Type of Parameter Passed	Number Passed	Size	Time (us)		
				I	O	I_O
p_n.a		0	0	9		
p_i_1.a	Integer	1		9	20	21
p_i_10.a	Integer	10		10	21	21
p_i_100.a	Integer	100		55	70	121
p_e_1.a	Enumeration	1		9	19	19
p_e_10.a	Enumeration	10		11	21	21
p_e_100.a	Enumeration	100		53	72	119
p_a_1.a	Array of Integer	1	1	10	21	21
p_a_10.a	Array of Integer	1	10	10	21	21
p_a_100.a	Array of Integer	1	100	11	22	22
p_a_10k.a	Array of Integer	1	10000	12	22	23
p_r_1.a	Record of Integer	1	1	10	21	21
p_r_100.a	Record of Integer	1	100	22	23	23
p_u_a_1.a	Unconstrained array	1	1	10	21	21
p_u_a_100.a	Unconstrained array	1	100	12	23	23
p_u_a_10k.a	Unconstrained array	1	10000	12	23	23
p_u_r_1.a	Unconstrained record	1	1	10	23	23
p_u_r_100.a	Unconstrained record	1	100	11	23	24

Observations about the results in Table 21 indicate:

1. The overhead for passing 100 integers is higher than the overhead for passing 1 integer (due to the time required for copying the integers on the stack when the procedure call is made).

2. The timings for records and arrays to indicate that they are passed by reference as opposed to by copy.

3. The timings for unconstrained types seem to suggest that there is very little extra overhead in passing the constraint information in the procedure call. Also, unconstrained records and arrays are passed by reference.

Instantiations of Generic Code: In the tests for inter- and intra-package calls, the subprograms are part of generic packages that are instantiated. These benchmarks measure additional overhead involved in executing generic instantiations of the code. Table 22 (for intra-package) and 23 (for inter-package) list the types of the parameters that are used in these tests. In Tables 22 and 23, the headings under the Time column: I, O, I_O have the times listed for parameters with mode **in, out,** and **in out** respectively.

TABLE 22. Subprogram Overhead (Intra-Package With generic Instantiation)

Execution time in microseconds

File Name	Type of Parameter Passed	Number Passed	Size	Time (us)		
				I	O	I_O
g_n.a		0	0	19		
g_i_1_c.a	Integer	1		12	24	25
g_i_10_c.a	Integer	10		23	34	35
g_i_100_c.a	Integer	100		56	79	121
g_e_1_c.a	Enumeration	1		12	22	24
g_e_10_c.a	Enumeration	10		24	33	34
g_e_100_c.a	Enumeration	100		54	89	119
g_a_1_c.a	Array of Integer	1	1	13	23	24
g_a_10_c.a	Array of Integer	1	10	13	25	23
g_a_100_c.a	Array of Integer	1	100	12	24	23
g_a_10k_c.a	Array of Integer	1	10000	13	24	24
g_r_1_c.a	Record of Integer	1	1	12	25	25
g_r_100_c.a	Record of Integer	1	100	23	26	26

Observations about the results in Table 22 indicate that:

1. The overhead for passing 100 integers is higher than the overhead for passing 1 integer (due to the time required for copying the integers on the stack when the procedure call is made).
2. The timings for arrays and records indicate pass by reference.
3. Generic instantiation procedure call is more expensive than non-generic instantiation procedure call.

TABLE 23. Subprogram Overhead (Inter-Package With Generic Instantiation)

Execution time in microseconds

File Name	Type of Parameter Passed	Number Passed	Size	Time (us)		
				I	O	I_O
c_n.a		0	0	18		
c_i_1.a	Integer	1		18	35	32
c_i_10.a	Integer	10		17	37	39
c_i_100.a	Integer	100		75	102	129
c_e_1.a	Enumeration	1		18	34	35
c_e_10.a	Enumeration	10		19	34	35
c_e_100.a	Enumeration	100		81	109	129
c_a_1.a	Array of Integer	1	1	18	36	36
c_a_10.a	Array of Integer	1	10	19	35	35
c_a_100.a	Array of Integer	1	100	20	34	34
c_a_10k.a	Array of Integer	1	10000	21	43	41
c_r_1.a	Record of Integer	1	1	19	35	33
c_r_100.a	Record of Integer	1	100	36	35	36

Observations about the results in Table 23:

1. Times as listed for inter-package reference with generic instantiations are much higher than with generic instantiations.

4.2.8 Pragmas

Table 23 lists the Pragma benchmarks. None of the pragma benchmarks (except for Pragma Shared) produced any response on execution. The compiler vendor has been contacted.

TABLE 24. Pragma Benchmarks

File Name	Benchmark Description	Time Difference (us)
pr00001.a	Pragma SUPPRESS used for Overflow_Check, Division_Check, and Range_Check	No response
pr00001_1.a	Pragma SUPPRESS used for Access_Check	No response
pr00001_2.a	Pragma SUPPRESS used for Index_Check and Length_Check	No response
pr00001_3.a	Pragma SUPPRESS used for STORAGE_CHECK	No response
pr00001_4.a	Pragma SUPPRESS used for ELABORATION_CHECK	No response
pr00001_5.a	Pragma SUPPRESS used for INDEX_CHECK	No response
pr00002.a	Pragma CONTROLLED used for access type	No response
pr00003.a	Pragma SHARED,shared integer updated	No effect
pr00003_1.a	Pragma SHARED,shared integer updated during rendezvous	No effect

Some observations about the results in Table 24 are:

1. From the results obtained and also from looking at the compiler documentation, it was determined that the DDC-I compiler does not implement pragma SHARED.

4.2.9 Input/Output

Table 25 lists the benchmarks that deal with TEXT_IO.

TABLE 25. Input/Output Benchmarks

Execution Time in Milliseconds

File Name	Benchmark Description	Time
io00001.a	Create output file and copy characters	2789.0
io00002.a	Create output file, copy data using ENUMERATION_IO	2167.0
io00003.a	Create output file, copy data using INTEGER_IO	2109.0
io00004.a	Create output file, copy data using FLOAT_IO	2009.0
io00005.a	Create output file, copy data using FIXED_IO	1978.0

These benchmarks have to be compared with the results of other compilers.

4.3 Runtime Implementation Benchmarks

4.3.1 Tasking Implementation Dependencies

Table 26 lists the benchmarks that determine tasking implementation dependencies.

TABLE 26. Tasking Implementation Benchmarks

File Name	Benchmark Description	Results
rt_t001.a	Is task space deallocated on return from procedure on task termination	Yes
rt_t002.a	Is task space deallocated upon task termination when access type is declared in library unit	No
rt_t003.a	Determine order of elaboration when several tasks are activated	See below
rt_t004.a	Can a task continue execution after its activation but prior to completion of activation of tasks declared in the same declarative part	Yes
rt_t005.a	If allocation of task raises STORAGE_ERROR when is exception raised	Task Activation
rt_t006.a	What happens to tasks declared in a library package when main task terminates	Do not terminate
rt_t007.a	Print default attribute STORAGE_SIZE and SIZE for tasks objects	0 bytes 16 bytes
rt_t008.a	Order of evaluation of tasks in abort statement	See below

Some observations from the results in Table 26 are:

1. For rt_t003.a, the results for the DDC-I compiler indicate that the tasks are activated in a random order and do not depend on the place where the task or its body is declared.
2. For rt_t007.a, the attribute TASK'STORAGE_SIZE was printed as 0. This is incorrect and the compiler vendor has been contacted with the results.
3. The DDC-I compiler aborts the tasks in the order they are named in the abort statement.

4.3.2 Task Synchronization

Table 27 lists the benchmarks that determine task synchronization implementation dependencies. The results for these benchmarks are listed below the table as the results cannot fit in the table column size.

TABLE 27. Rendezvous Implementation Benchmarks

File Name	Benchmark Description
rt_r001.a	Algorithm used when choosing among branches of selective wait statement
rt_r002.a	Order of evaluation of guard conditions in a selective wait
rt_r003.a	Method to select from delay alternatives of the same delay in selective wait
rt_r004.a	Determine when expressions of an open delay alternative or entry family index in an open accept alternative evaluated
rt_r005.a	Determine the priority of a task which has no explicit priority specified
rt_r006.a	Determine the priority of a rendezvous between two tasks which have no explicit priorities specified

1. rt_r001.a: The DDC-I compiler accepts the entry calls in the reverse order that they are declared in the select statement. This implies that real-time embedded programmers using the DDC-I compiler should place their most critical accept statements at the end of the select statement. If a program is designed using this knowledge, it may present performance and portability problems if the application changes the compiler for which the program was designed initially.
2. rt_r002.a: The DDC-I compiler evaluates the guard conditions in the reverse order that they are declared in the select statement.
3. rt_t003.a: The DDC-I compiler always selects the first delay alternative of the same delay in a selective wait.
4. rt_r004.a: This benchmark did not execute on the DDC-I compiler. The benchmark compiled fine, but could not execute. The program never returned. The compiler vendor has been contacted.

5. rt_r005.a: This benchmark determines the default priority of a task with undefined priority value. For the DDC-I compiler, the default priority of a task with undefined priority is PRIORITY'FIRST.

6. rt_r006.a: This benchmark determines the default priority of a rendezvous between two tasks with undefined priorities. For the DDC-I compiler, the default priority of a rendezvous between two tasks with undefined priorities is PRIORITY'FIRST.

4.3.3 Scheduling and Delay Statement

Table 28 lists the benchmarks for Scheduling and delay statement dependencies.

TABLE 28. Scheduling and Delay Statement Dependencies

File Name	Benchmark Description	Results
dt00001.a	Determine minimum delay time	0.001 sec
dt00002.a	Determine if user tasks are pre-emptive	Yes
dt00003.a	Determine method to share processor within each priority level	See below
dt00004.a	Does delay 0.0 cause scheduling	Yes

dt0003.a: For the DDC-I compiler, if time slicing is not enabled then tasks of the same priority execute to completion unless a synchronization point is reached.

4.3.4 Memory Management

Table 29 lists the benchmarks for memory management dependencies.

TABLE 29. Memory management Dependencies

File Name	Benchmark Description	Results
m00001.a	Determine STORAGE_ERROR threshold	376 k
m00002.a	Is Unchecked_Deallocation implemented	Yes
m00003.a m00003_1.a	Garbage Collection performed on fly	No
m00004.a	Garbage Collection performed on scope exit	No

Some observations about the results in Table 29 are:

1. For the system that these benchmarks were run on, 94 arrays of 1000 integers was the maximum storage space allocated. At this point STORAGE_ERROR was raised. The size of the memory space available is approximately 376 kilobytes.

4.3.5 Asynchronous I/O

rt_io001.a: The results of this benchmark show that the DDC-I compiler does not implement true asynchronous I/O.

4.4 Real-Time Paradigms

Real-time paradigms can be coded in Ada using macro constructs and benchmarked. Also, a compiler implementation may recognize these paradigms and perform optimizations to implement that paradigm much more efficiently.

Table 30 lists real-time paradigms that have been benchmarked.

TABLE 30. Real-time Paradigms

File Name	Benchmark Description	Time (us)
pa00001.a	Simple producer consumer transaction with main calling consumer task	108
pa00001_1.a	Simple producer consumer transaction with consumer using selective wait	168
pa00001_2.a	Simple producer consumer transaction with producer task calling consumer task	108
pa00001_3.a	producer task communicates with consumer task through a bounded buffer	356
pa00001_4.a	producer task communicates with consumer task indirectly through a bounded buffer with a transporter between buffer and consumer	461
pa00001_5.a	producer task communicates with consumer task indirectly through a bounded buffer with a transporter between buffer and producer as well as transporter between buffer and consumer	583
pa00001_6.a	Producer task communicates with a consumer via relay	217
pa00002.a	Monitor using semaphores	Error
pa00002_1.a	Monitor using rendezvous	604
pa00002_2.a	Monitor using rendezvous	Tasking_Error
pa00004.a	Abort a task and create a new one	3100

Some observations about the results in Table 30 are:

1. In pa00001.a, the time measured is the time it takes for the producer to call the entry in the consumer, the start of rendezvous with the consumer accepting the information, and the beginning of execution of the calling task. This is equivalent to two context switches: the first from the main task to the called task and the second from the called task to the main task. Time for this interaction is 108 microseconds.

2. In pa00001_1.a, the time measured is the time it takes for the producer to call the entry in the consumer, the start of rendezvous with the consumer accepting the information, and the beginning of execution of the calling task. Time for rendezvous in this case is 168 microseconds.

3. This is similar to previous test, except that a producer task calls an entry in the consumer task, instead of the main task calling an entry in the consumer task. Both the producer and consumer task have the highest priority possible (PRIORITY'LAST). Time for a single rendezvous in this case is 108 microseconds.

4. In pa00001_3.a, the producer task communicates with the consumer task indirectly through a bounded buffer. Time taken by the consumer to receive information from the producer via the buffer task is 356 microseconds.

5. In pa00001_4.a, a producer task communicates with a consumer task indirectly through a bounded buffer with a transporter between the buffer and the consumer. Time taken by the consumer to receive information from the producer via the buffer and transporter tasks is 461 microseconds.

6. In pa00001_5.a, a producer task communicates with a consumer task indirectly through a bounded buffer with a transporter between the buffer and the producer as well as between the buffer and the consumer. Time taken by the consumer to receive information from the producer is 583 microseconds.

7. In pa00001_6.a, a producer task communicates with a consumer via the relay. Time taken by the consumer to receive information from the producer is 217 microseconds.

5. Conclusions

In this report, real-time benchmarks developed at the Center for Software Engineering [1] have been run on the DDC-I Ada compiler system targeted to the Intel 80XXX family of microprocessors as well as on the self-hosted HP Ada compiler for the HP 9000/350 computer. Extensive measurements have been made in the areas of importance for real-time systems and the results analyzed. By running these benchmarks on additional compilers, the validity of using this set of Real-time benchmarks has been demonstrated and they have been proven to be portable to other environments. The results produced by running the real-time benchmarks provide valuable and useful information to programmers in evaluating compilers for real-time applications. The results of the Real-time benchmarks in Appendix A also enable users to compare the performance of different releases of the HP Ada compiler.

The performance of Ada systems is a complex subject and assessing and comparing performance for real-time programming is a difficult task. The goal of the Real-time benchmarks is to measure the performance of Ada features of importance for programming real-time systems as well the Ada RTS implementation dependencies. Although the benchmark programs are designed to avoid unnecessary and unwanted operating system interference, there are certain basic functions of the operating system that cannot be avoided in some cases (e.g. virtual memory support). It may also be the case that no operating system is present, and the compiler's RTS is responsible for supporting all runtime functions. No matter how the support of the runtime environment is implemented, the Real-time benchmarks are intended to measure the performance of what is actually available to the user. The Real-time benchmarks provide a set of benchmarks that are easily portable to other systems and can be used to determine the performance of Ada real-time features.

REFERENCES

[1] CECOM Center for Software Engineering, "Real-time Performance Benchmarks for Ada", C02-092LY-0001-00, Final Report delivered by Arvind Goel, March 1989.

[2] CECOM Center for Software Engineering, "Documentation for Real-time Performance Benchmarks for Ada", C02-092LY-0002-00, Final Report delivered by Arvind Goel, March 1989.

[3] R.M. Clapp et al., "Towards Real-time Performance Benchmarks for Ada", CACM, Vol. 29, No. 8, August 1986.

[4] N. Altman, "Factors Causing Unexpected Variations in Ada Benchmarks", Technical Report, CMU/SEI-87-TR-22, October 1987.

[5] N. Altman et al., "Timing Variation in Dual Loop Benchmarks" , Technical Report, CMU/SEI-87-TR-21, October 1987.

[6] CECOM Center for Software Engineering, "Catalog of Ada Runtime Implementation Dependencies", C02 092LA 0001, Final report delivered by LabTek (revised ARTEWG document), 15 Feb 1989.

[7] "Catalog of Interface Features and Options for the Ada Runtime Environment ", ARTEWG Report, December, 1987.

[8] M. D. Broido, "Toward Real-time Performance Benchmarks For Ada", Technical Correspondence, CACM, Vol. 30, No. 2, February 1987.

[9] L. MacLaren, "Evolving Toward Ada in Real-time Systems", Proceedings of the ACM, SIGPLAN Symposium on the Ada Programming Language, November, 1980.

[10] N. Weiderman et al., "Ada for Embedded Systems: Issues and Questions", Technical Report, CMU/SEI-87-TR-26, October 1987.

[11] SofTech Inc., "Real-time Ada", July, 1984.

[12] A. Goel, E.Wong, "Evaluation of Existing Benchmark Suites For Ada", Ada Technology Conference Proceedings, Washington DC, March 15-20,1988.

[13] CECOM Center for Software Engineering, "Establish and Evaluate Ada Runtime Features of Interest for Real-time Systems", C02092LA0003, Final report delivered by IITRI, 15 Feb 1989.

[14] "Proceedings of the International Workshop on Real-time Ada Issues", UK, 13-15 May, 1987, pages 10-11.

[15] A. Tetewsky, A. Clough, R. Racine, R. Whittredge, "Mapping Ada onto Embedded Systems:Memory Constraints", Ada Letters, September/October, 1988.

[16] Ada Compiler Evaluation Capability (ACEC) Technical Operating Report (TOR) User's Guide, Report D500-11790-2, Boeing Military Aerospaces, P.O. Box 7730, Wichita, Kansas, 1988.

[17] Ada Compiler Evaluation Capability (ACEC) Version Description Document, Report D500-11790-3, Boeing Military Airplane, P.O. Box 7730, Wichita, Kansas, 1988.

[18] CECOM Center For Software Engineering, "Performance Measurements of the CHS Ada Compiler", Final report delivered by Unixpros Inc., 15 December, 1989.

Appendix A: Execution Results For the HP Ada Compiler

Appendix A presents the results of running the Real-time benchmarks on the HP Ada Compiler (Releases 3.25 and 4.35) running on HP 9000/350 machine under HP-UX Release 6.2. The hardware and software configuration is as follows:

HP Testbed Hardware and Software:

The hardware used for benchmarking was Hewlett-Packard 9000/350 CPU running HP-UX V 6.2. The setup can be summarized as follows:

Host: **HP 9000/350 running HP-UX V 6.2.**

Compiler: **Self-hosted HP (basically the Alsys Ada Compiler)**
Ada Development System Version 3.25 and 4.35.

Target: **Same as the host.**

Tables 31 through 61 list the results of running the Real-time benchmarks on the HP Ada compilers. Any observations about the results are presented below the corresponding tables. A detailed description of the performance measurements of the HP Ada compilers is presented in report [18].

TABLE 31. HP Results: Task/Activation Termination Benchmarks

Execution time in Microseconds

File Name	Benchmark Description	HP3.25	HP4.35
t00001.a	Task type in main, object in block statement	14281.5	10079.2
t00001_1	Task object is declared directly in block statement	14178.0	11089.0
t00001_2.a	Task type and object defined in package procedure	14167.0	11010.0
t00001_3.a	Task type in package, object in package procedure	13989.0	10890.0
t00001_4.a	Task type and object are declared in another task	14267.0	11008.0
t00002.a	Task type and array elaborated in a procedure	14389.0	11890.0
t00002_1.a	Task type in package, array in procedure	14379.0	11870.0
t00002_2.a	Task type in main, array in package procedure	14567.0	12089.0
t00003.a	Task object is declared as part of record	13980.0	10789.0
t00004.a	Task access type in main, task created via new	14221.7	11682.2
t00004_1.a	Task access type in block, task created via new	14210.0	11689.0
t00004_2.a	Task access type in main, array created via new	14217.0	11790.0
t00005.a	Task object in block statement, idle tasks=1	14234.0	11678.0
t00005_1.a	Task object in block statement, idle tasks=5	14567.0	11980.0
t00005_2.a	Task object in block statement, idle tasks=10	14678.0	11897.0
t00005_3.a	Task object in block statement, idle tasks=20	14789.0	11789.0
t00006.a	Task created via new allocator, idle tasks=1	14220.0	11683.0
t00006_1.a	Task created via new allocator, idle tasks=5	14367.0	11768.0
t00006_2.a	Task created via new allocator, idle tasks=10	14456.0	11789.0
t00006_3.a	Task created via new allocator, idle tasks=20	14983.0	11834.0

- Results for the HP4.35 compiler show an improvement of nearly 4000 microseconds (30 %) for task activation/termination timings for tasks not allocated via the new allocator.
- Results for the HP4.35 show an improvement of nearly 2500 microseconds (18 %) for task activation/termination timings via the new operator.

- There is little effect on task activation/termination timings when the number of idle tasks increases.

TABLE 32. HP Results: Simple Rendezvous Benchmarks (No Parameters Passed)

Time in Microseconds

Name	Description	HP3.25	HP4.35
r00001.a	Procedure calls entry of task declared in main	1756.0	1702.0
r00001_1.a	Procedure calls entry in task created via new	1768.0	1697.0
r00001_2.a	Main calls entry in task decl in package	1778.0	1787.0
r00002.a	Main calls two entries in two tasks decl in package	1769.0	1690.0
r00002_1.a	Main calls 10 entries in ten tasks decl in package	1806.0	1702.0
r00002_2.a	Main calls 10 entries in one task decl in package	1799.0	1708.0
r00003	Main calls 1st entry in select, 2 entries declared	2612.0	2248.0
r00003_1	Main calls last entry in select, 2 entries declared	2656.0	2260.0
r00003_2.a	Main calls 1st entry in select, 10 entries decl	3389.0	2667.0
r00003_3.a	Main calls last entry in select, 10 entries decl	3399.0	2697.0
r00003_4.a	Main calls 6th entry in select, 10 entries decl	3395.0	2687.0
r00003_5	Main calls 1st entry in select, 20 entries declared	4022.0	3474.0
r00003_6	Main calls last entry in select, 20 entries declared	4058.0	3508.0
r00003_7	Main calls 11th entry in select, 20 entries declared	4024.0	3474.0
r00004.a	Main calls 1st entry out of 2, 1st guard true next false	2612.0	2249.0
r00004_1.a	Main calls last entry out of 2, 1st guard false next true	2665.0	2259.0
r00004_2.a	Main calls 1st entry out of 20, 1st guard true rest false	4021.0	3471.0
r00004_3.a	Main calls last entry out of 20, last guard true rest false	4059.0	3589.0
r00004_4.a	Main calls 11th entry out of 20, 11th guard true rest false	4032.0	3491.0
r00004_5.a	Main calls 11th entry out of 20, all guards true	4056.0	3678.0

- For both the HP compilers, the timing for rendezvous is nearly the same for the scenarios in which the main program calls ten entries in 10 different tasks or the main program calls 10 entries in one task.
- The more the number of entries in a select statement, the more time it takes to rendezvous with any entry in the select statement (for both the HP compilers).

- Position of the accept in the select statement does not affect the rendezvous timings for both HP compilers.

TABLE 33. HP Results: Complex Rendezvous Benchmarks

Time in Microseconds

Name	Direction Passed	Type and Number Passed	Size	HP3.25	HP4.35
r00005_i	In	Integer Array	1	2070.0	1838.0
r00005_o	Out	Integer Array	1	2080.0	1870.0
r00005_io	In Out	Integer Array	1	2080.0	1870.0
r00005_1_i	In	Integer Array	1000	1974.0	1908.0
r00005_1_o	Out	Integer Array	1000	1984.0	1924.0
r00005_1_io	In Out	Integer Array	1000	1984.0	1924.0
r00005_2_i	In	Integer Array	10000	1974.0	1908.0
r00005_2_o	Out	Integer Array	10000	1984.0	1924.0
r00005_2_io	In Out	Integer Array	10000	1984.0	1924.0
r00005_3_i	In	1 Integer		2062.0	1816.0
r00005_3_o	Out	1 Integer		2069.0	1820.0
r00005_3_io	In Out	1 Integer		2069.0	1820.0
r00005_4_i	In	10 Integers		2056.0	1818.0
r00005_4_o	In Out	10 Integers		2078.0	1834.0
r00005_4_io	In Out	10 Integers		2078.0	1834.0
r00005_5_i	In	100 Integers		2784.0	1930.0
r00005_5_o	Out	100 Integers		2765.0	1996.0
r00005_5_io	In Out	100 Integers		2765 0	1996.0

- Rendezvous times indicate that arrays are passed by reference for both HP compilers.
- As far as passing integer parameters during a rendezvous, the HP compilers use pass by copy (as the time for rendezvous increases with the increase in the number of integer parameters). Also, the rendezvous time for mode **in out** parameters is generally more than the time required for parameters of mode **in**.

- There is a significant improvement of 800 microseconds (30 %) for the HP4.35 compiler in rendezvous timings when 100 integer parameters are passed during the rendezvous.

TABLE 34. HP Results: More Rendezvous Benchmarks

Execution time in microseconds

File Name	Benchmark Description	HP3.25	HP4.35
r00006_1_1.a	1st entry out of 2 called with 10 integers	2089.0	1823.0
r00006_1_2.a	1st entry out of 2 called with 100 integers	2789.0	1978.0
r00006_2_1.a	Last entry out of 2 called with 10 integers	2378.0	1978.0
r00006_2_2.a	Last entry out of 2 called with 100 integers	2874.0	2456.0
r00006_3_1.a	1st entry out of 10 called with 10 integers	3387.0	2678.0
r00006_3_2.a	1st entry out of 10 called with 100 integers	3567.0	2890.0
r00006_4_1.a	Last entry out of 10 called with 10 integers	3434.0	2760.0
r00006_4_2.a	Last entry out of 10 called with 100 integers	3690.0	2867.0
r00006_5_1.a	1st entry out of 20 called with 10 integers	3645.0	2867.0
r00006_5_2.a	1st entry out of 20 called with 100 integers	3789.0	2899.0
r00006_6_1.a	Last entry out of 20 called with 10 integers	3900.0	2956.0
r00006_6_2.a	Last entry out of 20 called with 100 integers	3989.0	2978.0
r00007.a	Overhead due to terminate alternative	28.9	148.3
r00008.a	Overhead of conditional entry call,rendezvous complete	5.0	4.8
r00008_1.a	Overhead of conditional entry call,rendezvous incomplete	25.0	21.0
r00009.a	Overhead of timed entry call,rendezvous complete	5.9	4.9
r00009_1.a	Overhead of timed entry call,rendezvous incomplete	27.0	20.4
r00011.a	Main calls an entry with 100 Integers,Idle tasks = 1	2784.0	1978.0
r00011_1.a	Main calls entry with 100 Integers,Idle tasks = 5	2876.0	1989.0
r00011_2.a	Main calls entry with 100 Integers,Idle tasks = 10	2877.0	1976.0
r00011_3.a	Main calls entry with 100 Integers,Idle tasks = 20	2901.0	1999.0

TABLE 35. HP Results: Memory Allocation: Storage Allocated is Fixed

Time in Microseconds

Name	Type Declared	Number Declared	Size	HP3.25	HP4.35
dd_in1	Integer	1		0.8	0.2
dd_in10	Integer	10		5.8	2.6
dd_in100	Integer	100		21.6	20.0
dd_st1	String	1	1	1.0	1.0
dd_st10	String	1	10	1.6	1.0
dd_st100	String	1	100	3.2	2.2
dd_en1	Enumeration	1		0.8	0.2
dd_en10	Enumeration	10		5.8	2.6
dd_en100	Enumeration	100		21.6	20.0
dd_ar1	Array of Integer	1	1	0.6	0.5
dd_ar10	Array of Integer	1	10	0.6	0.5
dd_ar100	Array of Integer	1	100	0.6	0.6
dd_ar1k	Array of Integer	1	1000	0.6	0.6
dd_ar10k	Array of Integer	1	10000	0.6	0.6
dd_ar100k	Array of Integer	1	100000	0.6	0.6

- For the HP Ada compilers, time required to allocate integer variables, enumeration variables, strings and arrays of integers upon entering a subprogram is negligible.

TABLE 36. HP Results: Dynamic Memory Allocation: Storage Allocated is Variable

Time in Microseconds

Name	Type Declared	Number Declared	Size of Object	HP3.25	HP4.35
dd_1d1	1-D Dynamically Bounded Array	1	1	10.4	9.8
dd_1d10	1-D Dynamically Bounded Array	1	10	10.6	10.2
dd_2d1	2-D Dynamically Bounded Array	1	1	15.8	15.0
dd_2d10	2-D Dynamically Bounded Array	1	100	15.6	14.8
dd_3d1	3-D Dynamically Bounded Array	1	1	25.6	24.4
dd_3d1C	3-D Dynamically Bounded Array	1	1000	25.6	24.6

- Time required for allocating dynamically bounded arrays increased with the number of dimensions for the HP Ada compilers.

TABLE 37. HP Results: Dynamic Memory Allocation with the NEW Allocator

Time in Microseconds

Name	Type Declared	Size of Object	HP3.25	HP4.35
dn_in1	Integer	1	200.0	80.0
dn_en1	Enumeration	1	200.0	80.0
dn_st1	String	1	140.0	80.0
dn_st10	String	10	140.0	80.0
dn_st100	String	100	260.0	80.0
dn_ar1	Integer Array	1	140.0	80.0
dn_ar10	Integer Array	10	160.0	80.0
dn_ar100	Integer Array	100	220.0	80.0
dn_ar1k	Integer Array	1000	260.0	80.0
dn_rc1	Integer Record	1	140.0	80.0
dn_rc10	Integer Record	10	160.0	80.0
dn_rc100	Integer Record	100	220.0	80.0
dn_1d1	1-D Dynamically Bounded Array	1	220.0	80.0
dn_1d10	1-D Dynamically Bounded Array	10	280.0	100.0
dn_2d1	2-D Dynamically Bounded Array	1	260.0	100.0
dn_2d10	2-D Dynamically Bounded Array	100	280.0	100.0
dn_3d1	3-D Dynamically Bounded Array	1	260.0	120.0
dn_3d10	3-D Dynamically Bounded Array	1000	280.0	120.0

- HP4.35 compiler takes at least 40% less time for memory allocation/deallocation via the new allocator as compared to HP3.25 compiler.

TABLE 38. HP Results: New Allocator: No Storage Deallocation

Time in Microseconds

Name	Type Declared	Size of Object	HP3.25	HP4.35
dn_in1	Integer	1	210.0	85.0
dn_en1	Enumeration	1	210.0	85.0
dn_st1	String	1	146.0	86.0
dn_st10	String	10	146.0	84.0
dn_st100	String	100	264.0	83.0
dn_ar1	Integer Array	1	149.0	87.0
dn_ar10	Integer Array	10	164.0	81.0
dn_ar100	Integer Array	100	223.0	82.0
dn_ar1k	Integer Array	1000	264.0	85.0
dn_rc1	Integer Record	1	139.0	84.0
dn_rc10	Integer Record	10	161.0	89.0
dn_rc100	Integer Record	100	220.0	80.0
dn_1d1	1-D Dynamically Bounded Array	1	223.0	89.0
dn_1d10	1-D Dynamically Bounded Array	10	286.0	110.0
dn_2d1	2-D Dynamically Bounded Array	1	263.0	110.0
dn_2d10	2-D Dynamically Bounded Array	100	279.0	104.0
dn_3d1	3-D Dynamically Bounded Array	1	264.0	129.0
dn_3d10	3-D Dynamically Bounded Array	1000	281.0	121.0

TABLE 39. HP Results: NEW Allocator:Active Tasks = 5

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	HP3.25	HP4.35
dn_st100.a	String	1	100	260.0	80.0
dn_ar1k.a	Integer Array	1	1000	260.0	80.0
dn_rc100.a	Record of Integer	1	100	220.0	80.0
dn_1d10.a	1-D Dynamically Bounded Array	1	10	280.0	100.0
dn_2d10.a	2-D Dynamically Bounded Array	1	100	280.0	100.0
dn_3d10.a	3-D Dynamically Bounded Array	1	1000	280.0	120.0

TABLE 40. HP Results: NEW Allocator:Active Tasks = 10

Execution time in microseconds

File Name	Type Declared	Number Declared	Size of Object	HP3.25	HP4.35
dn_st100.a	String	1	100	260.0	80.0
dn_ar1k.a	Integer Array	1	1000	260.0	80.0
dn_rc100.a	Record of Integer	1	100	220.0	80.0
dn_1d10.a	1-D Dynamically Bounded Array	1	10	280.0	100.0
dn_2d10.a	2-D Dynamically Bounded Array	1	100	280.0	100.0
dn_3d10.a	3-D Dynamically Bounded Array	1	1000	280.0	120.0

- Results show negligible effect of idle tasks on memory allocation/deallocation timings.

TABLE 41. HP Results: Exception Raised/Handled in Block - HP3.25

Execution time in microseconds

File Name	Exception not raised	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00001.a	1.2	2134.5	2189.0	2234.0	2349.0	2136.0
e00001_1.a	1.3	2144.5	2199.0	2254.0	2379.0	2186.0
e00001_2.a	1.3	2149.5	2191.0	2258.0	2389.0	2189.0

TABLE 42. HP Results: Exception Raised/Handled in Block - HP4.35

Execution time in microseconds

File Name	Exception not raised	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00001.a	1.2	1407.6	1456.0	1501.0	1489.0	1478.0
e00001_1.a	1.2	1427.6	1476.0	1511.0	1499.0	1488.0
e00001_2.a	1.2	1437.6	1479.0	1515.0	1491.0	1489.0

- HP4.35 shown an improvement of about 25% over the HP3.25 compiler for exception handling timings.

TABLE 43. HP Results: Exception Raised/Handled One Level Above - HP3.25

Execution time in microseconds

File Name	Exception not raised	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00001.a	1.2	2934.5	2989.0	3034.0	3249.0	3036.0
e00001_1.a	1.3	3044.5	3099.0	3054.0	3279.0	3086.0
e00001_2.a	1.3	3049.5	3091.0	3058.0	3289.0	3089.0

- Time to propagate an exception one level above is ~950 microseconds for the HP3.25 compiler.

TABLE 44. HP Results: Exception Raised/Handled One Level Above - HP4.35

Execution time in microseconds

File Name	Exception not raised	User defined explicit	Constraint _error explicit	Constraint _error implicit	Numeric _error explicit	Numeric _error implicit
e00001.a	1.2	2207.6	2256.0	2301.0	2289.0	2278.0
e00001_1.a	1.2	2227.6	2276.0	2311.0	2299.0	2288.0
e00001_2.a	1.2	2237.6	2279.0	2315.0	2291.0	2289.0

- Time to propagate an exception one level above is ~850 microseconds for the HP4.35 compiler.

TABLE 45. HP Results: Tasking_Error Exception Benchmarks

Execution time in microseconds

File Name	Benchmark Description	HP3.25	HP4.35
e00005.a	Exception Raised in rendezvous,0 idle tasks	8331.0	5308.0
e00005_1.a	Exception Raised in rendezvous,5 idle tasks	8333.0	5408.0
e00005_2.a	Exception Raised in rendezvous,10 idle tasks	8343.0	5309.0
e00006.a	Child task has error during elaboration,0 idle tasks	8456.0	5345.0
e00006_1.a	Child task has error during elaboration,5 idle tasks	8467.0	5347.0
e00006_2.a	Child task has error during elaboration,10 idle tasks	8499.0	5434.0

TABLE 46. HP Results: Chapter 13 Benchmarks

Execution time in microseconds

File Name	Benchmark Description	HP3.25	HP4.35
h00001.a	Boolean operations on arrays,Pragma P .CK	14.5	4.7
h00001_1.a	Boolean operations on arrays,Rep Clause	14.9	5.6
h00001_2.a	Boolean operations on arrays,not packed	15.6	15.9
h00002.a	Boolean operations on array components,Pragma Pack	23.0	7.9
h00002_1.a	Boolean operations on array components,Rep Clause	26.9	8.1
h00002_2.a	Boolean operations on array components,not packed	11.9	7.8
h00003.a	Assignment,comparison on arrays of booleans,Pragma PACK	10.3	4.6
h00003_1.a	Assignment,comparison on boolean arrays,Rep Clause	11.5	4.9
h00003_2.a	Assignment,comparison on boolean arrays,not packed	17.9	7.9
h00004.a	Assign,compare whole records,no rep clause	8.2	8.7
h00004_1.a	Assign, compare whole records,rep clause	8.9	4.5
h00004_2.a	Assign, compare whole records,Pragma PACK	9.6	6.7
h00005.a	Unchecked_conversion, Integer object to another	0.9	0.8
h00005_1.a	Unchecked_conversion, String to Integer	1.2	1.1
h00005_2.a	Unchecked_conversion,Floating array to record	1.9	2.1
h00006.a	Store, extract record bit fields, no rep clause	11.2	7.6
h00006_1.a	Store, extract record bit fields, rep clause	13.4	9.7
h00006_2.a	Store, extract record bit fields, rep clause	14.2	10.1
h00008.a	Store, extract record bit fields defined by nested rep clauses using packed arrays	13.8	8.1
h00009.a	Change of representation from one record to another	12.1	4.8
h00010.a	POS,SUCC, and PRED operations on enum type with rep clause numbered with gaps in internal coding	19.1	13.2
h00010_1.a	POS,SUCC, and PRED operations on enum type with rep clause numbered with no gaps in internal coding	23.1	11.2

TABLE 47. HP Results: CLOCK Function Tests

Execution time in microseconds

File Name	Benchmark Description	HP3.25	HP4.35
c00001.a	CLOCK function overhead	1134.3	1220.1
c00002.a	CLOCK resolution	100.0	100.0

TABLE 48. HP Results: TIME and DURATION Mathematics

Execution time in microseconds

File Name	Operation Performed	HP3.25	HP4.35
tm1.a	Time = Var_time + Var_duration	2.3	2.3
tm2.a	Time = Var_time + Const_duration	2.7	2.9
tm3.a	Time = Var_duration + Var_time	2.2	2.1
tm4.a	Time = Const_duration + Var_time	2.3	2.9
tm5.a	Time = Var_time - Var_duration	2.7	2.0
tm6.a	Time = Var_time - Const_duration	2.3	2.1
tm7.a	Duration = Var_time - Var_time	2.4	2.3
tm8.a	Duration = Var_duration + var_duration	2.4	2.1
tm9.a	Duration = Var_duration + Const_duration	2.5	2.2
tm10.a	Duration = Const_duration + Var_duration	2.4	2.1
tm12.a	Duration = Const_duration + Const_duration	2.6	2.2
tm12.a	Duration = Var_duration - Var_duration	2.5	2.1
tm13.a	Duration = Var_duration - Const_duration	2.4	2.1
tm14.a	Duration = Const_duration - Var_duration	2.5	2.2
tm15.a	Duration = Const_duration - Const_duration	2.6	2.2

TABLE 49. HP Results: Numeric Computation Benchmarks

Execution time in microseconds

File Name	Operation Performed	HP3.25	HP4.35
tm16.a	Float Matrix Multiplication	995.0	845.0
tm17.a	Float Matrix Addition	901.4	924.0
tm18.a	Factorial Calculation	105.0	139.0
tm19.a	Square root calculation	456.0	398.0

TABLE 50. HP Results: Subprogram Overhead (Intra-Package)

Time in Microseconds

Name	Parameter Type	Number Passed	Size	HP3.25			HP4.35		
				In	Out	In Out	In	Out	In Out
d_n		0		2.0			2.0		
d_i_1	Integer	1		4.0	5.0	6.0	4.0	5.0	6.0
d_i_10	Integer	10		8.0	10.0	12.0	8.0	10.0	12.0
d_i_100	Integer	100		62.0	66.0	118.0	64.0	68.0	124.0
d_a_1	Array of Integer	1	1	2.0	2.0	2.0	2.0	2.0	2.0
d_a_10	Array of Integer	1	10	2.0	2.0	2.0	2.0	2.0	2.0
d_a_100	Array of Integer	1	100	4.0	6.0	6.0	6.0	6.0	8.0
d_a_10k	Array of Integer	1	10000	2.0	2.0	2.0	2.0	2.0	2.0
d_u_a_1	Unconstrained array	1	1	2.0	4.0	4.0	4.0	4.0	2.0
d_u_a_100	Unconstrained array	1	100	2.0	2.0	2.0	4.0	4.0	4.0
d_u_a_10k	Unconstrained array	1	10000	4.0	2.0	4.0	4.0	4.0	4.0

- For parameters of type integer, subprogram overhead for variables of mode **out** and **in out** is greater than that of mode **in**. This is because of the additional overhead involved in copying back the parameters of mode **out** and **in out** when returning from the procedure call. However, the overhead for passing 100 integers of mode *in out* is twice as much (124 microseconds) for 100 integers of mode *out* for both the HF compilers.
- The overhead for passing 100 integers is higher than the overhead for passing 1 integer (due to the time required for copying the integers on the stack when the procedure call is made).
- The timings for arrays of integer indicate that it is passed by reference. (as pass by reference times do not vary with the length of the array passed).
- The timings for unconstrained array types suggest that there is very little extra overhead in passing the constraint information in the procedure call.

TABLE 51. HP Results: Subprogram Overhead (Intra-Package with Pragma INLINE)

Time in Microseconds

Name	Parameter Type	Number Passed	Size	HP3.25			HP4.35		
				In	Out	In Out	In	Out	In Out
i_n		0		2.0			2.1		
i_i_1	Integer	1		2.0	2.0	2.0	2.0	2.0	2.0
i_i_10	Integer	10		6.1	8.2	12.1	6.0	8.0	12.0
i_i_100	Integer	100		61.0	65.0	122.0	60.0	66.0	120.0
i_a_1	Array of Integer	1	1	1.9	2.1	2.2	2.0	2.0	2.1
i_a_10	Array of Integer	1	10	1.8	2.1	2.1	2.0	2.0	2.0
i_a_100	Array of Integer	1	100	1.9	2.0	2.1	2.0	2.0	2.0
i_a_10k	Array of Integer	1	10000	2.0	2.1	2.1	2.0	2.0	2.0
i_u_a_1	Unconstrained array	1	1	2.9	2.8	2.8	2.7	2.7	2.7
i_u_a_100	Unconstrained array	1	100	2.9	2.8	2.8	2.7	2.6	2.7
i_u_a_10k	Unconstrained array	1	10000	2.7	2.8	2.9	2.6	2.7	2.7

- The timings indicate that procedure calls with pragma INLINE execute faster than procedures without pragma INLINE.

TABLE 52. HP Results: Subprogram Overhead (Inter-Package)

Time in Microseconds

Name	Parameter Type	Number Passed	Size	HP3.25			HP4.35		
				In	Out	In Out	In	Out	In Out
p_n		0		2.0			2.0		
p_i_1	Integer	1		4.0	4.0	6.0	4.0	4.0	6.0
p_i_10	Integer	10		8.0	10.0	12.0	8.0	6.0	12.0
p_i_100	Integer	100		62.0	56.0	118.0	64.0	60.0	124.0
p_a_1	Array of Integer	1	1	2.0	2.0	2.0	2.0	2.0	2.0
p_a_10	Array of Integer	1	10	2.0	2.0	2.0	2.0	2.0	2.0
p_a_100	Array of Integer	1	100	4.0	6.0	6.0	6.0	6.0	8.0
p_a_10k	Array of Integer	1	10000	2.0	2.0	2.0	2.0	2.0	2.0
p_u_a_1	Unconstrained array	1	1	2.0	4.0	4.0	4.0	4.0	2.0
p_u_a_100	Unconstrained array	1	100	2.0	2.0	2.0	4.0	4.0	4.0
p_u_a_10k	Unconstrained array	1	10000	4.0	2.0	4.0	4.0	4.0	4.0

- The results indicate that there is no extra overhead between inter-package subprogram call as opposed to intra-package subprogram calls.

TABLE 53. HP Results: Subprogram Overhead (Intra-Package with Generic Instantiations)

Time in Microseconds

Name	Parameter Type	Number Passed	Size	HP3.25			HP4.35		
				In	Out	In Out	In	Out	In Out
g_n_c		0		2.0		2.0			
g_i_1_c	Integer	1		4.0	4.0	6.0	4.0	4.0	5.0
g_i_10_c	Integer	10		8.0	8.0	18.0	8.0	8.0	18.0
g_i_100_c	Integer	100		60.0	68.0	130.0	66.0	68.0	114.0
g_a_1_c	Array of Integer	1	1	6.0	8.0	8.0	6.0	8.0	8.0
g_a_10_c	Array of Integer	1	10	8.0	10.0	10.0	8.0	10.0	10.0
g_a_100_c	Array of Integer	1	100	12.0	14.0	15.0	11.0	13.0	14.0
g_a_10k_c	Array of Integer	1	10000	12.0	14.0	14.0	12.0	14.0	14.0

- For integer parameters, the timings are compatible for intra-package with generic instantiations as opposed to without generic instantiations.
- For arrays of integers, there is an increase of more than 50% for intra-package with generic instantiations as opposed to without generic instantiations.

TABLE 54. HP Results: Subprogram Overhead (Inter-Package with Generic Instantiations)

Time in Microseconds

Name	Parameter Type	Number	Size	HP3.25			HP4.35		
				In	Out	In Out	In	Out	In Out
c_n		0		2.0		2.0			
c_i_1	Integer	1		4.0	6.0	6.0	2.0	4.0	6.0
c_i_10	Integer	10		8.0	8.0	18.0	8.0	8.0	18.0
c_i_100	Integer	100		60.0	68.0	132.0	75.0	79.0	119.0
c_a_1	Array of Integer	1	1	10.0	10.0	14.0	4.0	6.0	6.0
c_a_10	Array of Integer	1	10	22.0	26.0	48.0	12.0	14.0	20.0
c_a_100	Array of Integer	1	100	220.0	236.0	408.0	72.0	80.0	144.0
c_a_10k	Array of Integer	1	10000	23362.0	23958.0	49526.0	9360.0	9556.0	20744.0

- For integer arrays of size 100 or more, there is a big difference in the times without generic instantiations and with generic instantiations: 49526 (*20744*) microseconds for the HP3.25 (*HP4.35*) compiler with generic instantiation as opposed to 2.0 (*2.0*) microseconds without generic instantiation for an array of 10000 integers (with mode *in out*). This indicates that inter-package calls with generic instantiation are extremely inefficient on the HP compilers as opposed to inter-package calls without generic instantiation.
- The HP4.35 compiler has an improvement of more than 100% for these timings as opposed to the HP3.25 compiler.

TABLE 55. HP Results: Pragma Benchmarks

File Name	Benchmark Description	HP3.25	HP4.35
pr00001.a	Pragma SUPPRESS used for Overflow_Check, Division_Check, and Range_Check	19.1	20.3
pr00001_1.a	Pragma SUPPRESS used for Access_Check	12.3	13.4
pr00001_2.a	Pragma SUPPRESS used for Index_Check and Length_Check	32.9	29.8
pr00001_3.a	Pragma SUPPRESS used for STORAGE_CHECK	3.4	3.9
pr00001_4.a	Pragma SUPPRESS used for ELABORATION_CHECK	No response	
pr00001_5.a	Pragma SUPPRESS used for INDEX_CHECK	26.4	21.9
pr00002.a	Pragma CONTROLLED used for access type	No effect	No effect
pr00003.a	Pragma SHARED,shared integer updated	No effect	No effect
pr00003_1.a	Pragma SHARED,shared integer updated during rendezvous	No effect	No effect

- The results indicate that Pragma SUPPRESS improves the execution timings for both the HP compilers.
- Pragms CONTROLLED and SHARED are not implemented for the HP Ada compilers.

TABLE 56. HP Results: Input/Output Benchmarks

Execution Time in Milliseconds

File Name	Benchmark Description	HP3.25	HP4.35
io00001.a	Create output file and copy characters	2169.0	2121.0
io00002.a	Create output file, copy data using ENUMERATION_IO	2245.0	2189.0
io00003.a	Create output file, copy data using INTEGER_IO	2345.0	2235.0
io00004.a	Create output file, copy data using FLOAT_IO	2509.0	2470.0
io00005.a	Create output file, copy data using FIXED_IO	2457.0	2346.0

TABLE 57. HP Results: Tasking Implementation Benchmarks

File Name	Benchmark Description	HP3.25	HP4.35
rt_t001.a	Is task space deallocated on return from procedure on task termination	Yes	Yes
rt_t002.a	Is task space deallocated upon task termination when access type is declared in library unit	Yes	Yes
rt_t003.a	Determine order of elaboration when several tasks are activated	See below	See below
rt_t004.a	Can a task continue execution after its activation but prior to completion of activation of tasks declared in the same declarative part	Yes	Yes
rt_t005.a	If allocation of task raises STORAGE_ERROR when is exception raised	Task Activation	Task Activation
rt_t006.a	What happens to tasks declared in a library package when main task terminates	Do not terminate	Do not terminate
rt_t007.a	Print default attribute STORAGE_SIZE and SIZE for tasks objects	16916 bytes 16 bytes	11796 bytes 16 bytes
rt_t008.a	Order of evaluation of tasks in abort statement	See below	See below

rt_t003.a There is no set order of elaboration when several tasks are activated for both the HP3.25 and HP4.35 Ada compilers.

rt_t008.a Tasks are aborted in the order that they are named in the abort statement for both the HP compilers.

TABLE 58. HP Results: Rendezvous Implementation Benchmarks

File Name	Benchmark Description
rt_r001.a	Choosing among branches of selective wait statement
rt_r002.a	Evaluation of guard conditions in a selective wait
rt_r003.a	Method to select from delay alternatives of the same delay in selective wait
rt_r004.a	When expressions of an open delay alternative or entry family index in an open accept alternative are evaluated
rt_r005.a	Priority of a task with no explicit priority specified
rt_r006.a	Priority of a rendezvous between two tasks which have no explicit priorities specified

rt_r001.a: No set algorithm for both the HP compilers.

rt_r002.a: Evaluation occurs in the order that they are declared in the select statement for both HP compilers.

rt_r003.a: No set algorithm for both the HP compilers.

rt_r004.a: The HP compilers always evaluates the expressions of an open delay statement before a selection is made.

rt_r005.a Lowest priority for both the HP compilers.

rt_r006.a Lowest priority for both the HP compilers.

TABLE 59. HP Results: Scheduling and Delay Statement Dependencies

File Name	Benchmark Description	HP3.25	HP4.35
dt00001.a	Determine minimum delay time	0.02 sec	0.02 sec
dt00002.a	Determine if user tasks are pre-emptive	Yes	Yes
dt00003.a	Determine method to share processor within each priority level	See below	See below
dt00004.a	Does delay 0.0 cause scheduling	No	No

dt00003.a: The scheduling facility in the Ada runtime is priority-based. If a task is available (queued) to run at a certain priority, the scheduler will choose it over tasks of lower priority when the scheduler is entered. The scheduler is entered whenever a task is to be descheduled. A task can be descheduled for any of the following reasons:

- completion of its time slice
- waiting for a rendezvous
- executing a *delay* statement
- waiting for the completion of I/O
- a higher priority task becomes schedulable because a delay statement expires.

In addition, at a variety of points at which the excution of a task requires entry into the Ada runtime, the Ada runtime itself checks to see that the currently-scheduled task is still the task of highest available priority; if not, the task will be descheduled. The points at which the scheduler checks for a higher priority task include rendezvous, delay, I/O, heap operations, and (because they engender heap operations) some string operations.

TABLE 60. HP Results: Memory Management Dependencies

File Name	Benchmark Description	HP3.25	HP4.35
m00002.a	Is Unchecked_Deallocation implemented	Yes	Yes
m00003.a m00003_1.a	Garbage Collection performed on fly	No	No
m00004.a	Garbage Collection performed on scope exit	No	No

TABLE 61. HP Results: Real-time Paradigms

File Name	Benchmark Description	HP3.25	HP4.35
pa00001.a	Simple producer consumer transaction with main calling consumer task	1879.0	1796.0
pa00001_1.a	Simple producer consumer transaction with consumer using selective wait	2310.0	2109.0
pa00001_2.a	Simple producer consumer transaction with producer task calling consumer task	1867.0	1782.0
pa00001_3.a	producer task communicates with consumer task through a bounded buffer	5170.0	4867.0
pa00001_4.a	producer task communicates with consumer task indirectly through a bounded buffer with a transporter between buffer and consumer	6423.0	6123.0
pa00001_5.a	producer task communicates with consumer task indirectly through a bounded buffer with a transporter between buffer and producer as well as transporter between buffer and consumer	8534.0	8167.0
pa00001_6.a	Producer task communicates with a consumer via relay	3345.0	3196.0
pa00002.a	Monitor using semaphores	2786.0	2421.0
pa00002_1.a	Monitor using rendezvous	9198.0	9002.0
pa00002_2.a	Monitor using rendezvous	1956.0	1867.0
pa00004.a	Abort a task and create a new one	27879.0	21834.0